Object-Oriented Analysis and Design

- Analysis vs. Design
- Analysis Activities
- Finding the Objects/ Classes
- An Analysis Example
- The Unified Modeling Language

Early 90's "Method Wars"

- Explosion of OO methods/notations
- No agreed terminology
- No standards
- No method/notation fully satisfied users needs
- Booch, OMT and OOSE are the most popular methods

➔ New generations of methods/notations
➔ Unification efforts

History of UML

- Rational hired the "three amigos"
  - Booch and Rumbaugh (10/94)
  - ➔ *Unified Method* (UML v0.8), 10/95
  - Jacobson (fall '95)
  - ➔ Focus on language (instead of method)
- UML partners consortium in '96
- UML v1.0, 1/97
- OMG standard (UML v1.1, 11/97)
- ISO standard (UML v1.3, 10/00)
- UML 2.0 adopted by OMG (draft 6/03)
- Latest update 7/05 (still UML 2.0)

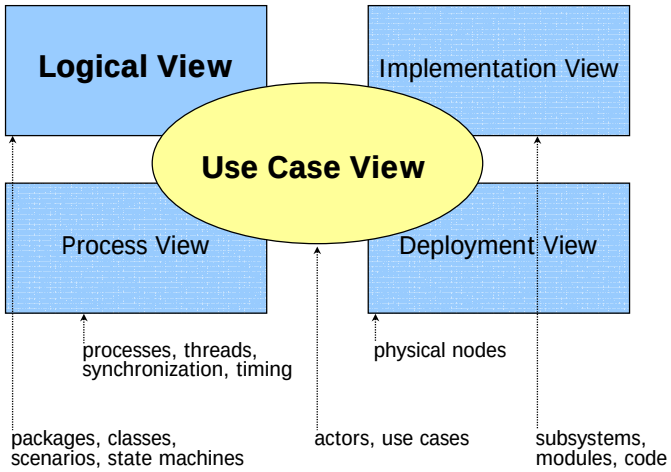
UML 2.0

- **Infrastructure**: UML kernel and extension mechanisms
- **Superstructure**: UML user level constructs
- **OCL**: Object Constraint Language
- **Diagram Interchange**

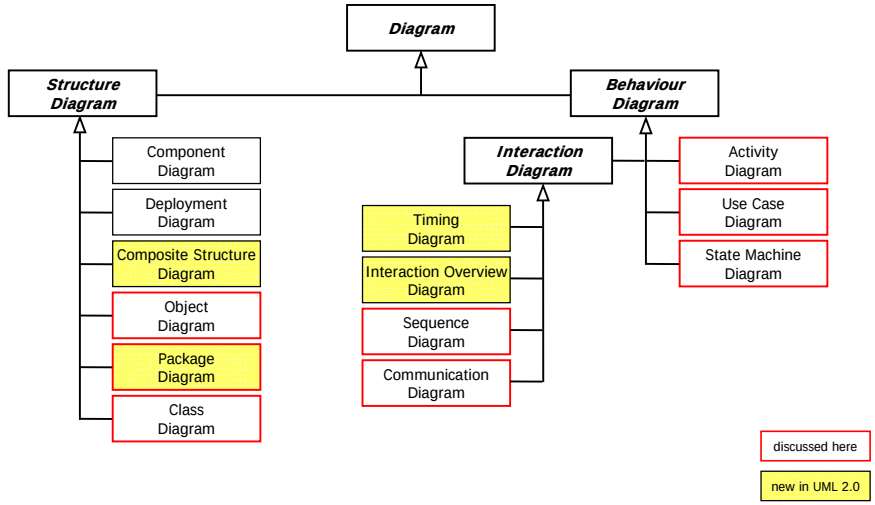
  Currently diagram interchange is supported by **XMI** (**X**ML **M**etadata **I**nterchange) a standardised XML-DTD for UML, developed by an IBM-lead consortium

- The **UML metamodel** describes the UML model elements
  - Abstract syntax in terms of UML
  - Well-formedness rules in terms of OCL and English prose
  - Semantics (model usage) in terms of English prose

Rational's 4+1 View

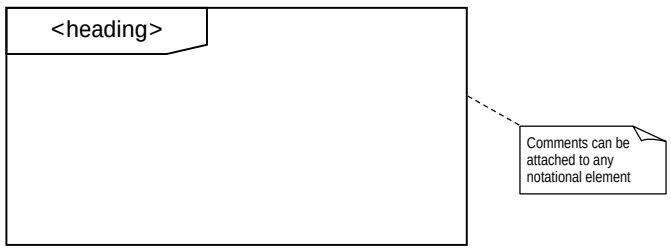


UML 2.0 Diagrams



Name Spaces

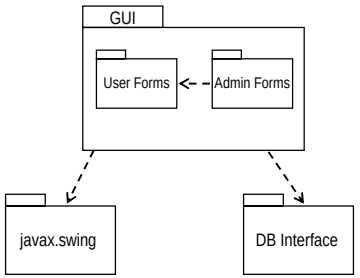
- UML diagrams may have a frame and heading
- The heading represents kind, name, and parameters of the enclosing namespace or "owner"



heading :: [<kind>] name [parameters]
kind :: activity | class | component | interaction (sd) | package | state machine | use case

Package Diagrams

- Grouping of classes and packages
- Defines namespaces á la C++

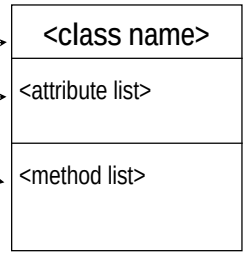


- Inofficially used in UML1.x for all diagram types

Class Diagrams

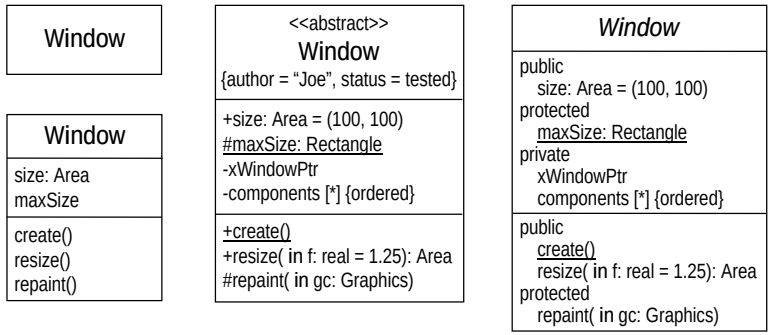
- Static analysis and design models
- Types of objects and their (static) interrelationships
- Basically similar to UML1.x (but more carefully defined)
- Main elements:
  - Classes
  - Attributes
  - Methods
  - Relationships
    - Dependency
    - Association
    - Aggregation/Composition
    - Generalisation
    - Realisation

strength



Class Notations

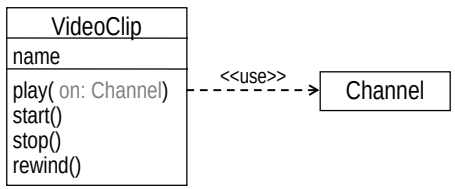
- Various presentation options



Attribute :: [visibility] [/] name [: type] [multiplicity] [= default-value] [{ property-string }]
Operation :: [visibility] name ([parameter-list]) [: return-type [{ property-string }]]
Parameter :: [direction] name : type-expression [multiplicity] [= default-value] [{ property-string }]

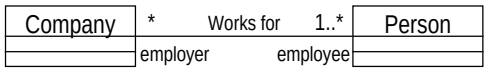
Dependency

- One model element requires another for specification or implementation
- A change in the required element may affect the dependent
- Dependencies can have names (there is a set of predefined dependency keywords ⇨ *stereotypes*)

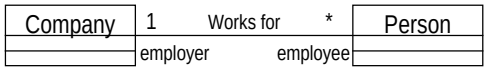


Association 1

- Defines a set of links between objects
- Associations can be navigated (in both directions by default)
- Associations are quite similar to the relationships known from ER modelling
- Associations have cardinalities
- Associated classes may have roles



Implementing Associations

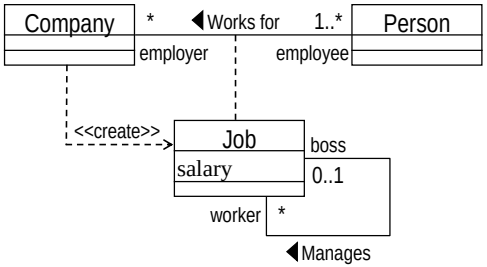



```
public class Company {
    private Person [ ]  employee;
}
```

```
public class Person {
    private Company employer;
}
```

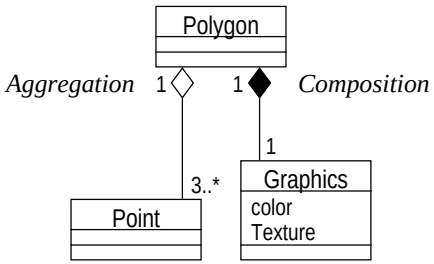

More Associations

- Associations may have attributes and behaviour (*association classes*)
- Association classes are quite similar to the relationship types known from ER modelling
- Association names may have reading directions



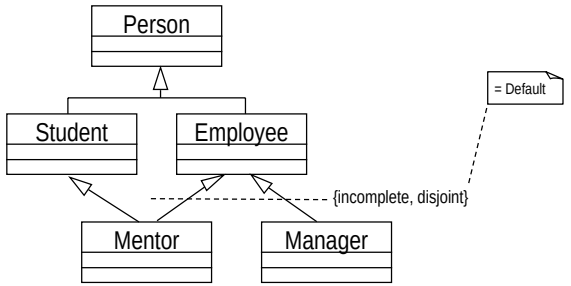
Aggregation and Composition

- Denote the *part-of* or *has* relationship
- Composition is the stronger form
  - Parts are not shared
  - Parts are existence dependent on the whole

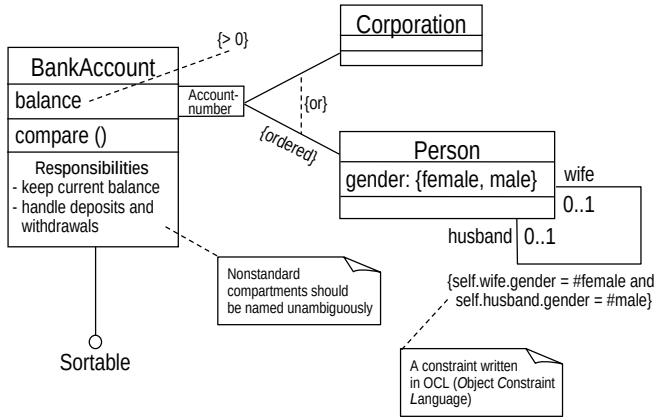


Generalisation

- Generalisation specifies the *is-a* relationship
- Strongest relationship between classes
- Subclasses inherit properties from superclasses
- Shown as class hierarchies



Notes, Constraints, Qualifiers, and Interfaces

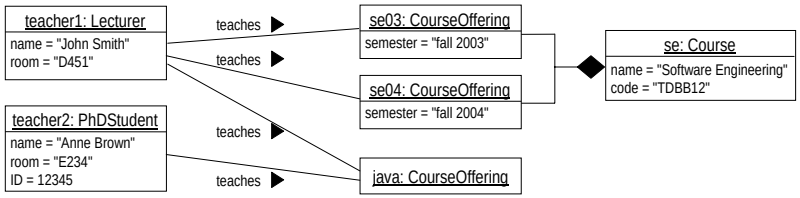


Some Guidelines for Class Diagrams

- Keep the analysis model simple
- Many simple classes are better than a few complex ones
- Use meaningful and familiar names
- Avoid “god classes,” behaviour should be distributed among classes
- Use multiple inheritance very sparely
- Try to avoid the too general dependency
- Associate methods with the providers of a service, i.e. the “responsible” classes
- Name all associations
- Document your model carefully

Object Diagrams

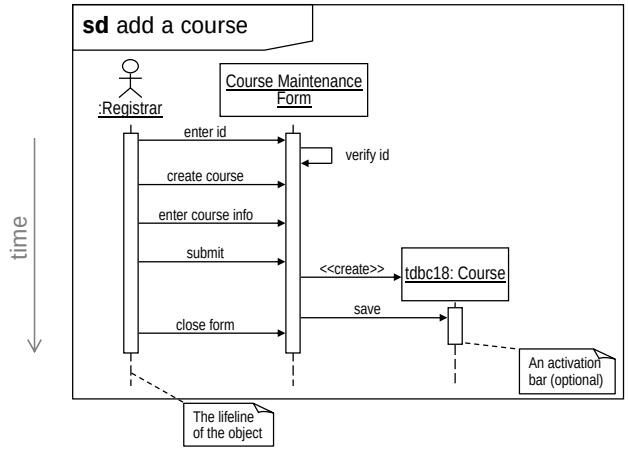
- Snapshot of system state
- Shows instances of interest
- Good to analyse complex object structures
- Unchanged from UML 1.x



Sequence Diagrams

- Graphical notation for scenarios
- Focus on interactions and time
- Describe how objects collaborate to provide a service
  - Ordered message flow
  - Object creation/destruction
  - ...
- Good tool to uncover missing behaviour and/or missing objects/classes
- Much more advanced in UML 2.0
- Communication diagrams are semantically equivalent, but focus on structural relationships between objects

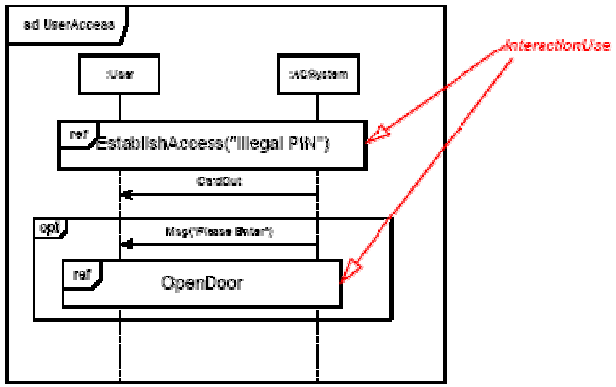
A Simple Sequence Diagram



Sequence Diagram Details

- Interaction fragments
  - References to other (interaction) diagrams (**ref**)
  - Conditionals (**alt**, **else**, **opt**) and loops (**loop**)
  - ...
- Notations for concurrency
  - Parallel fragments (**par**)
  - Asynchronous messages
- Lots of special annotations
  - State invariants
  - Timing and duration constraints
- Quite some changes from UML 1.x

Another Sequence Diagram



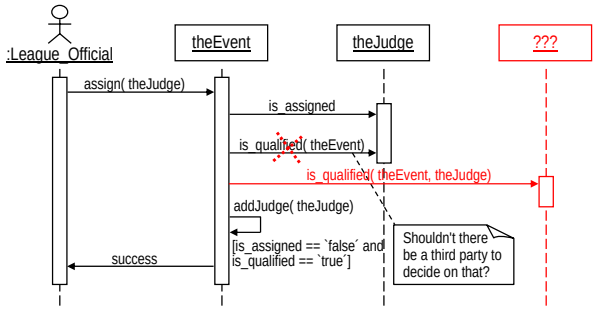
See sect. 14 in UML Superstructure Specification (05-07-04)

Some Guidelines for Sequence Diagrams

- Keep the analysis scenarios simple
- Analysis scenarios should usually be initiated by an actor
- Discriminate instances of the same class
- Make all assumption clear, use pre- and postconditions
- Concentrate on the major scenarios
- Manage consistency between use cases, scenarios, and class diagrams
- Focus on general behaviour, do not implement methods too early (in UML 2.0 you could actually do that)

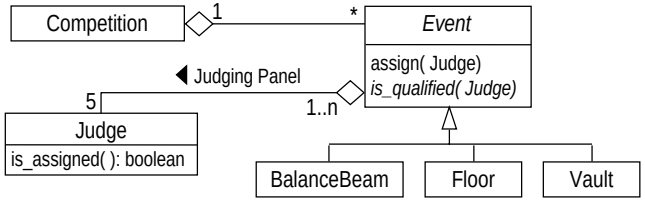
The GSS Case Study Continued 1

- Assign a judge to an event
  - An event has a judging panels assigned to it
  - The judging panel consists of “qualified scorers for this event”
- How could this work?



The GSS Case Study Continued 2

- Alternative 1: Let Event handle qualified scorers
  - Make Event an abstract class
  - Defer determination of qualified scorers to specific event classes

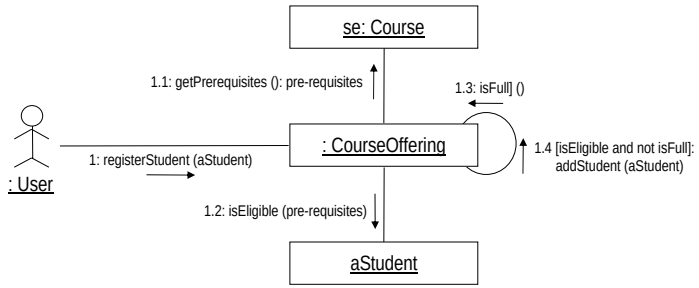


The GSS Case Study Continued 3

- Alternative 2..n:
  - Try to give other existing classes this responsibility
    - League
    - Equipment
    - ...
  - Define a special purpose class
  - Find out where this responsibility lies in the “real world”

Communication Diagrams

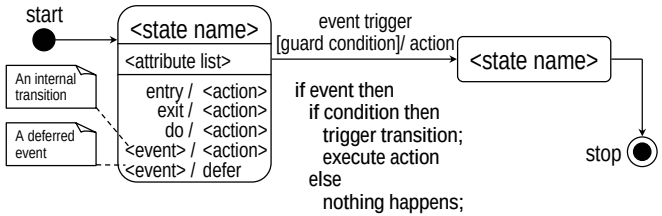
- Focus on interactions and links between objects
- Roughly equivalent to sequence diagrams



➔ Use RPDs instead (more expressive)

State Machine Diagrams 1

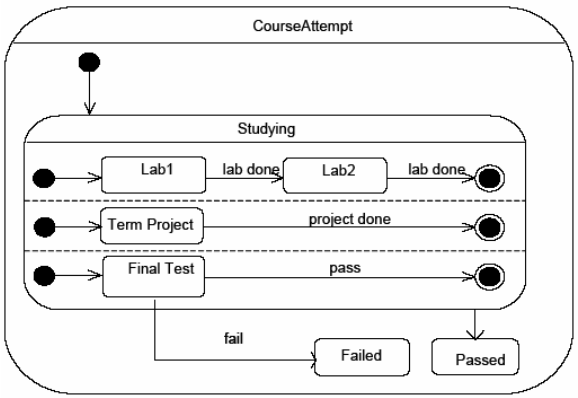
- Graphical notation for class behaviour modelling
- Describe all possible states and state changes triggered by external stimuli
- Good tool to describe complex object life cycles
- Good tool to uncover missing state information and behaviour



State Machine Diagrams 2

- States can be nested
- Substates may be concurrent
- Transitions may have multiple sources and/or destinations
- Formalised many informal parts from UML1.x
- Activity diagrams (see earlier lecture) are also state machines (see also here)

Another State Machine



See sect. 15 in UML Superstructure Specification (05-07-04)

Some Guidelines for Statechart Diagrams

- Use STDs when there is (complex) state- dependent behaviour
- Use STDs when you are not sure when things can (are allowed to) happen
- Ensure that all events are covered
- Ensure that all states are reachable
- Ensure that all states can be exited
- Ensure that all transitions can be triggered
- Check each pair of states for missing transactions
- Check consistency with other models

The CRS Case Study Again

- Course Registration System
- Adapted from Rose tutorial
- Textual analysis
  - Problem statement
  - Use cases
- Iterations over class diagram
- Some design considerations

➔ *See materials produced during the lecture*